Programacion

None

None

None

programacion

1. Bienvenido al módulo Programación

Hola

2. Unidad 1: Fundamentos de la Programación en Java

2.0.1 **Duración estimada:** 24 horas (3 semanas)

2.0.2 **Resultado de Aprendizaje (RA1):** Reconoce la estructura de un programa informático, identificando y relacionando los elementos propios del lenguaje de programación utilizado.

2.1 Semana 1: Los Ladrillos del Código: Variables, Datos y Constantes

2.1.1 PLAN DE TRABAJO (8 horas)

- **Sesión 1 (Teoría, 2 horas):** Variables, tipos de datos primitivos, constantes y literales.
- **Sesión 2 (Práctica, 3 horas):** La Ficha de Datos del Evento.
- **Sesión 3 (Práctica, 3 horas):** Valores Inmutables y Literales.

2.1.2 TEORÍA - SESIÓN 1

(Página 1 de 4)

2.1.3 1.- Variables, Datos y Constantes

¡Bienvenidos de nuevo al desarrollo de `EventDEV`! En el módulo de "Entornos de Desarrollo" montasteis vuestro taller y creasteis el esqueleto del proyecto. Ahora, en "Programación", vamos a empezar a construir con los ladrillos fundamentales de cualquier software: los datos.

Un programa necesita manejar información que puede cambiar: el nombre de un evento, el número de asistentes, el precio de una entrada... Para ello, la almacena en **variables**.

Una **variable** es un espacio con nombre que reservamos en la memoria del ordenador para guardar un dato. El nombre que le damos se llama **identificador** y nos permite acceder y modificar ese dato.

Caso práctico

Laia Claramunt reúne a Alba y a Pau. "Equipo, el siguiente paso en `EventDEV` es definir la información básica de un evento. Necesitamos una forma de almacenar su nombre, aforo, precio, si es benéfico y una categoría. En Java, esto se hace con variables, pero hay una regla clave: debemos declarar de antemano qué **tipo de dato** guardará cada variable. Hoy aprenderéis a elegir el tipo correcto para cada pieza de información."

1.1.- Creando Variables: Declaración e Inicialización

Antes de usar una variable, debemos crearla. Este proceso tiene dos pasos: 1. **Declaración:** Anunciar el tipo de dato y el nombre de la variable. `tipoDeDato nombreDeVariable;` 2. **Inicialización:** Asignar un valor inicial con el operador `=`. `nombreDeVariable = valor;`

Es una práctica común y recomendada realizar ambos pasos en una sola línea para asegurar que las variables siempre tengan un valor conocido desde el principio.

`tipoDeDato nombreDeVariable = valor;`

```
// Declaración e inicialización combinadas
String nombreEvento = "Concierto de Rock";
int numeroDeAsistentes = 150;
double precioEntrada = 75.50;
```

```
char categoriaEvento = 'C'; // C de Concierto
boolean estaAgotado = false;
```

Buenas Prácticas: Nombrando Variables

- **Usa `camelCase`:** La primera palabra en minúscula, las siguientes en mayúscula (`precioEntradaVip`).
- **Sé descriptivo:** `nombreUsuario` es mejor que `nu`. El código debe ser auto-explicativo.
- **Evita abreviaturas confusas.**

(Página 2 de 4)

2.1.4 2.- Tipos de Datos Primitivos

Java, para ser eficiente, proporciona un conjunto de ocho tipos de datos básicos, conocidos como **tipos primitivos**. Son la materia prima con la que representaremos toda la información. Se dividen en cuatro grupos principales.

2.1.- Tipos Numéricos Enteros

Para números sin decimales. La elección depende del rango de valores que necesitemos.

Tipo	Tamaño	Rango de Valores Aproximado	Uso Típico en `EventDEV`
`byte`	8 bits	-128 a 127	Número de ponentes en una charla.
`short`	16 bits	-32,768 a 32,767	Asientos en una sala pequeña.
`int`	32 bits	-2 mil millones a 2 mil millones	**El estándar:** aforo, ID de usuario.
`long`	64 bits	-9 trillones a 9 trillones	ID de una transacción bancaria.

`int` **es el tipo entero por defecto y el más utilizado** por su eficiencia en los procesadores modernos. Úsalo a menos que tengas una razón específica para necesitar un rango mayor (`long`) o para ahorrar memoria en grandes colecciones de datos (`byte`).

2.2.- Tipos Numéricos en Coma Flotante (Decimales)

Para números con decimales. La diferencia principal es la precisión (número de decimales exactos).

Tipo	Tamaño	Precisión Aproximada	Uso Típico en `EventDEV`
`float`	32 bits	~7 dígitos decimales	Coordenadas, cálculos gráficos.
`double`	64 bits	~15 dígitos decimales	**El estándar:** precios, valoraciones.

`double` **es el tipo decimal por defecto**. Su mayor precisión lo hace más seguro para la mayoría de los cálculos, especialmente los financieros.

2.3.- El Tipo Carácter: `char`

Almacena un **único carácter Unicode**, que puede ser una letra, un número o un símbolo. Se escribe entre **comillas simples (`'`)**. Es ideal para códigos de categoría, iniciales o respuestas de tipo 'S'/'N'.

`char categoria = 'A'; // 'A' para Adultos` `char simboloMoneda = '€';`

2.4.- El Tipo Lógico: `boolean`

Es el tipo más simple. Solo puede tener dos valores: `true` (verdadero) o `false` (falso). Es la base de toda la lógica y toma de decisiones en un programa.

`boolean esBenefico = true;` `boolean entradasDisponibles = false;`

Para Saber Más: El `String` no es primitivo

El tipo `String`, que usamos para texto (`"Concierto de Rock"`), **no es un tipo primitivo**. Es un **objeto**, una estructura más compleja que aprenderemos a manejar en profundidad más adelante. Por ahora, es suficiente saber que se usa para guardar cualquier cadena de texto entre **comillas dobles (`"`)**.

(Página 3 de 4)

2.1.5 3.- Literales y Constantes

3.1.- Literales: Valores Fijos en el Código

Un **literal** es un valor que escribimos directamente en nuestro código fuente. Cuando haces `int aforo = 500;`, `500` es un literal entero. Comprender su sintaxis es clave para evitar errores.

Tipo de Dato	Ejemplo de Literal	Notas
`int`	`123`	El tipo entero por defecto.
`long`	`9876543210L`	**Obligatorio** el sufijo `L` (o `l`).
`double`	`85.99`	El tipo decimal por defecto.
`float`	`4.5F`	**Obligatorio** el sufijo `F` (o `f`).
`char`	`'A'`	Siempre con comillas simples.
`boolean`	`true`	Palabra clave reservada (`true` o `false`).

Un error muy común es olvidar la `F` al asignar un valor decimal a una variable `float`. El compilador protestará porque, por defecto, interpreta el literal decimal como `double` y no cabe en un `float` sin una conversión explícita.

`float valoracion = 4.7; // ¡ERROR DE COMPILACIÓN!` `float valoracion = 4.7F; // CORRECTO`

3.2.- Constantes: Variables que No Deben Cambiar

Hay valores en un programa que son fijos por naturaleza, como el valor de PI o el porcentaje de IVA. Si usamos una variable normal, corremos el riesgo de modificarla por error, causando bugs. Para evitarlo, usamos **constantes**.

Una constante es una "variable de solo lectura", cuyo valor no puede ser modificado después de su inicialización. Se declara usando la palabra clave **`final`**.

`final tipoDeDato NOMBRE_DE_LA_CONSTANTE = valor;`

Convenciones para Constantes: 1. **Usa `final`:** Siempre. 2. **Nómbralas en `UPPER_SNAKE_CASE`:** Todo en mayúsculas, separando palabras con guion bajo. Esto las hace instantáneamente reconocibles.

```
// Constante para el IVA cultural que se aplicará en EventDEV
final double IVA_CULTURAL = 0.10;

// Constante para el número máximo de entradas que un usuario puede comprar
final int MAX_ENTRADAS_POR_USUARIO = 4;

// Esta línea, si la escribiéramos, provocaría un error de compilación
// MAX_ENTRADAS_POR_USUARIO = 5; // ERROR: Cannot assign a value to final variable
```

El uso de constantes aporta tres grandes ventajas: * **Seguridad:** Evita modificaciones accidentales. * **Legibilidad:** `precio * IVA_CULTURAL` es mucho más claro que `precio * 0.10`. * **Mantenibilidad:** Si el IVA cambia, solo tienes que modificarlo en un único lugar: la declaración de la constante.

(Página 4 de 4)

2.1.6 RESUMEN DE LA SEMANA 1 Y AUTOEVALUACIÓN

En esta primera sesión teórica, hemos sentado las bases para trabajar con datos en Java. Has aprendido los conceptos fundamentales que utilizarás en cada programa que escribas a partir de ahora.

Conceptos Clave: * **Variable:** Un espacio en memoria con nombre para guardar un dato que puede cambiar. Se debe **declarar** (indicar tipo y nombre) e **inicializar** (dar un valor). * **Tipos Primitivos:** Los ocho tipos de datos básicos de Java (`int`, `double`, `boolean`, `char`, etc.) que son la base para representar toda la información. * **Literal:** Un valor fijo escrito directamente en el código (ej. `100`, `"Hola"`, `true`). Recuerda los sufijos `L` para `long` y `F` para `float`. * **Constante:** Una variable declarada con `final` cuyo valor no puede cambiar. Es una buena práctica para valores fijos, mejorando la seguridad, legibilidad y mantenimiento del código.

Diagrama de Flujo Conceptual:

[DIAGRAMA: Un esquema simple. "Necesito guardar un dato" -> "¿Cambiará su valor?" (Sí -> Variable, No -> Constante (`final`)) -> "Elegir el tipo primitivo adecuado (`int`, `double`...)" -> "Declarar e inicializar" -> "Dato almacenado en memoria".]

Cita para pensar

"Los programas, en su esencia, son máquinas que transforman datos. Elegir la representación correcta para esos datos es el primer paso, y el más importante, para construir una máquina que funcione bien." - Anónimo

Autoevaluación

1. Elige el tipo de dato primitivo más apropiado para cada uno de los siguientes datos del proyecto `EventDEV`:
   - a) El número de valoraciones que ha recibido un evento.
   - b) Si un evento requiere autorización paterna o no.
   - c) El precio exacto de los gastos de gestión (ej: 2.50 €).
   - d) El código de una zona en un estadio (ej: 'F').
2. ¿Cuál es la diferencia principal entre una variable y una constante?
3. Escribe la línea de código para declarar una constante entera llamada `AFORO_MAXIMO_LEGAL` con el valor 1000.

(Respuestas: 1. a) `int` (o `long` si se esperan millones), b) `boolean`, c) `double`, d) `char`. 2. El valor de una variable puede cambiar, el de una constante (`final`) no. 3. `final int AFORO_MAXIMO_LEGAL = 1000;`)

2.1.7 PRÁCTICA - SESIÓN 2

(Página 1 de 2)

2.1.8 La Ficha de Datos del Evento

Caso práctico

Laia se acerca a Pau. —Bien, Pau. Ya tienes la teoría fresca. Ahora vamos a aplicarla. Abre el proyecto `EventDEV` que creaste en el otro módulo. Vamos a borrar el `println` de bienvenida y, en su lugar, vamos a definir la "ficha técnica" de un evento de prueba. Crearemos una variable para cada dato fundamental, eligiendo el tipo correcto, y luego las mostraremos por la consola. Es el primer paso para dar vida a nuestra aplicación.

Objetivos de la Sesión

Al finalizar esta práctica, serás capaz de: * Abrir y modificar un proyecto existente en IntelliJ IDEA. * Declarar, inicializar y utilizar variables de los principales tipos primitivos. * Concatenar cadenas de texto con variables para generar una salida formateada. * Resolver problemas de forma autónoma aplicando los conceptos de variables y tipos.

Desarrollo Paso a Paso (Parte Guiada)

1. **Abre tu proyecto** `EventDEV` en IntelliJ IDEA.
2. **Localiza y limpia el método** `main`**:** Navega hasta `App.java` y borra la línea `System.out.println(...)` que contiene.
3. **Declara e inicializa las variables del evento** dentro del `main`:

```
// --- FICHA DE DATOS DEL EVENTO ---
String nombreEvento = "Concierto de Leyendas del Rock";
int aforo = 5000;
double precioEntrada = 75.50;
char categoria = 'M'; // M: Musical, D: Deportivo, C: Cultural
boolean esBenefico = false;
```

4. **Muestra los datos por consola** usando concatenación:

```
// --- MOSTRANDO LOS DATOS POR PANTALLA ---
System.out.println("--- Ficha del Evento en EventDEV ---");
System.out.println("Nombre: " + nombreEvento);
System.out.println("Aforo: " + aforo + " personas");
System.out.println("Precio: " + precioEntrada + " EUR");
System.out.println("Categoría: " + categoria);
System.out.println("Es benéfico: " + esBenefico);
System.out.println("-----------------------------------");
```

5. **Ejecuta el programa** (clic en el triángulo verde o `Shift + F10`) y verifica que la salida es la esperada.

Experimenta y Aprende

Antes de pasar a los retos, dedica unos minutos a "romper" el código para entender mejor cómo funciona Java. Prueba lo siguiente y observa los errores que te muestra IntelliJ: * ¿Qué pasa si intentas asignar un texto a una variable `int`? (`aforo = "muchagente";`) * ¿Y si usas comillas simples para un `String`? (`nombreEvento = 'Concierto';`) * ¿Qué error da si olvidas el punto y coma al final de una línea?

(Página 2 de 2)

¡Ahora tú! (Retos)

Ahora que has completado la parte guiada, resuelve los siguientes retos creando nuevos proyectos o modificando el actual.

Reto 1: Ficha de Usuario (Consolidación) Crea un nuevo programa (o añade el código al final del `main` actual) para definir y mostrar una **ficha de usuario** para la app `EventDEV`. Debe incluir las siguientes variables: * Nombre de usuario (`String`), ej: "AlbaT". * Edad (`int`), ej: 28. * Saldo en la cuenta (`double`), ej: 250.75. * Inicial del nombre (`char`), ej: 'A'. * Si es un usuario VIP o no (`boolean`), ej: `true`.

Imprime todos estos datos por consola de forma clara y ordenada, similar a la ficha del evento.

Reto 2: Mensaje de Bienvenida Personalizado (Ampliación) Crea un tercer programa. Usando variables de la ficha de evento y de la ficha de usuario que ya has creado, construye y muestra por pantalla una única `String` que contenga un mensaje de bienvenida personalizado. El mensaje debe ser similar a este:

`"¡Hola AlbaT! Te informamos que el evento 'Concierto de Leyendas del Rock' (categoría M) ya está disponible. Como usuario VIP, tienes acceso preferente."`

Pista: Necesitarás concatenar múltiples variables y literales de texto en una sola instrucción `System.out.println()`.

Autoevaluación

- ¿He modificado el proyecto `EventDEV` sin problemas? **[SÍ / NO]**
- ¿He resuelto el Reto 1, eligiendo los tipos de datos correctos para el usuario? **[SÍ / NO]**
- ¿He conseguido construir el mensaje personalizado del Reto 2? **[SÍ / NO]**
- ¿Entiendo por qué Java da errores cuando intento asignar un tipo de dato incorrecto a una variable? **[SÍ / NO]**

2.1.9 **PRÁCTICA - SESIÓN 3**

(Página 1 de 2)

2.1.10 **Valores Inmutables: Constantes y Literales**

Caso práctico

Laia revisa el código de Alba: `double precioFinal = precioBase * 1.10;`. —Alba, esto funciona, pero podemos mejorarlo —dice Laia —. Ese `1.10` es un "número mágico". Declárala como una constante `IVA_CULTURAL`. Hará el código más legible y fácil de mantener si el IVA cambia en el futuro.

Objetivos de la Sesión

Al finalizar esta práctica, serás capaz de: * Declarar y utilizar constantes (`final`) para mejorar la legibilidad y mantenibilidad del código. * Comprender el concepto de inmutabilidad y el error de compilación asociado. * Utilizar correctamente los sufijos de literales (`L` y `F`) para `long` y `float`. * Resolver un escenario de depuración relacionado con tipos y constantes.

Desarrollo Paso a Paso (Parte Guiada)

1. **Abre tu proyecto** `EventDEV`.
2. **Declara constantes de negocio** dentro del método `main`:

```
final double IVA_CULTURAL = 0.10;
final int AFORO_MAXIMO_LEGAL = 50000;
```

3. **Usa las constantes en cálculos:**

```
double precioBaseEntrada = 45.0;
double precioFinalEntrada = precioBaseEntrada + (precioBaseEntrada * IVA_CULTURAL);
System.out.println("Precio Final con IVA: " + precioFinalEntrada + " EUR");
```

4. **Comprueba la inmutabilidad:** Escribe la línea `AFORO_MAXIMO_LEGAL = 60000;`. Observa el error que reporta IntelliJ: "**Cannot assign a value to final variable...**". Bórrala para continuar.
5. **Practica con literales** `long` **y** `float`**:**

```
long idTransaccion = 1234567890123L;
float valoracionMediaEvento = 4.7F;
System.out.println("ID de la última transacción: " + idTransaccion);
System.out.println("Valoración media del evento: " + valoracionMediaEvento);
```

6. **Ejecuta el programa** y verifica que la salida es la correcta.

Experimenta y Aprende

- Quita el sufijo `L` de la variable `idTransaccion`. Si el número es suficientemente grande, IntelliJ te dará un error: "Integer number too large". ¿Por qué? Porque sin la `L`, Java intenta tratarlo como un `int`, y el número no cabe.
- Quita el sufijo `F` de la variable `valoracionMediaEvento`. IntelliJ te dará un error: "incompatible types: possible lossy conversion from double to float". ¿Por qué? Porque sin la `F`, Java lo trata como `double`, que es más grande que `float`.

(Página 2 de 2)

¡Ahora tú! (Retos)

Reto 1: Calculadora Geométrica (Consolidación) Crea un nuevo programa que calcule el área y el perímetro de una circunferencia, que podría representar el radio de acción de un evento. 1. Declara una constante `double` para el número **PI** (usa `3.14159`). 2. Declara una variable `double` para el **radio** (dale el valor que quieras). 3. Crea una expresión para calcular el **área** (`PI * radio * radio`). 4. Crea una expresión para calcular el **perímetro** (`2 * PI * radio`). 5. Muestra los resultados por consola, indicando claramente qué es cada valor.

Reto 2: Depuración de Código (Ampliación) Copia el siguiente fragmento de código en un nuevo proyecto. Contiene varios errores relacionados con los conceptos de esta sesión. Usa los mensajes de error de IntelliJ para encontrar y corregir todos los bugs hasta que el programa se ejecute correctamente.

```
// CÓDIGO CON ERRORES
public class DepuracionTipos {
    public static void main(String[] args) {

        final String NOMBRE_APP = "EventDEV";
        int version = 1;

        System.out.println("Iniciando la versión " + version + " de " + NOMBRE_APP);

        // Se intenta actualizar la versión, pero es una constante
        NOMBRE_APP = "EventDEV 2.0";

        // Se declara un float sin el sufijo 'F'
        float costeDesarrollo = 1500.50;

        // Se declara un long que no cabe en un int sin el sufijo 'L'
        long lineasDeCodigo = 4000000000;

        System.out.println("Coste: " + costeDesarrollo);
        System.out.println("Líneas de código: " + lineasDeCodigo);
    }
}
```

Autoevaluación

- ¿Entiendo por qué y cómo se utiliza la palabra clave `final`? **[SÍ / NO]**
- ¿He resuelto el Reto 1 usando constantes y variables correctamente? **[SÍ / NO]**
- ¿He sido capaz de encontrar y corregir todos los errores del Reto 2? **[SÍ / NO]**
- ¿Sé cuándo y por qué debo usar los sufijos `L` y `F` en los literales numéricos? **[SÍ / NO]**

2.2 Semana 2: Dando Vida a los Datos: Operaciones y Expresiones

2.2.1 PLAN DE TRABAJO (8 horas)

- **Sesión 4 (Teoría, 2 horas):** Operadores (aritméticos, de asignación, relacionales, lógicos) y precedencia.
- **Sesión 5 (Práctica, 3 horas):** La Calculadora de Tickets (uso de operadores aritméticos y relacionales).
- **Sesión 6 (Práctica, 3 horas):** El Validador de Acceso (uso de operadores lógicos).

2.2.2 TEORÍA - SESIÓN 4

(Página 1 de 4)

2.2.3 3.- Operadores y Expresiones: La Lógica en Acción

En la primera semana aprendimos a crear "cajas" (variables) y a guardar datos en ellas. Pero un programa que solo guarda datos es como una calculadora sin botones de operaciones: inútil. La verdadera potencia de la programación reside en la capacidad de **manipular** esos datos: realizar cálculos, compararlos y tomar decisiones basadas en ellos. Para ello, utilizamos **operadores**.

Un **operador** es un símbolo especial que le indica al compilador que realice una manipulación matemática o lógica específica. Los valores, variables o constantes sobre los que actúa un operador se llaman **operandos**.

```
int recaudacionTotal = precioEntrada * numeroDeAsistentes;
```

En esta línea, `*` es el operador de multiplicación, y `precioEntrada` y `numeroDeAsistentes` son los operandos.

Una **expresión** es una combinación de operandos y operadores que, al ser evaluada por Java, produce un único valor como resultado. En el ejemplo anterior, `precioEntrada * numeroDeAsistentes` es una expresión, y su resultado es el valor que se asignará a `recaudacionTotal`.

Caso práctico

Alba y Pau están diseñando la lógica para la venta de entradas en `EventDEV`. —Vale, ya tenemos el precio de la entrada y la cantidad que quiere el usuario guardados en variables —dice Pau—. Ahora necesitamos calcular el subtotal. Eso es fácil, ¿no? Solo hay que multiplicar. —Correcto —responde Alba—. Usaremos el operador aritmético `*`. La expresión será `precioUnitario * cantidad`. Pero la cosa se complica. Laia nos ha pedido que apliquemos un descuento si el cliente es VIP **y** la compra supera los 100€. —Ah, eso ya no es un solo cálculo —reflexiona Pau—. Primero, necesitamos una forma de comprobar si el cliente es VIP. Eso será una expresión que nos devuelva `true` o `false`. Luego, otra para comprobar si la compra supera los 100€. Y finalmente, una forma de unir esas dos comprobaciones para que solo se cumpla si ambas son verdad. —Exacto. Para eso no nos valen los operadores aritméticos. Necesitaremos operadores relacionales para comparar y operadores lógicos para combinar. Hoy vamos a montar ese puzzle.

3.1.- Operadores Aritméticos

Son los que ya conoces de las matemáticas y se usan con tipos de datos numéricos.

Operador	Nombre	Ejemplo	Resultado
`+`	Suma	`10 + 5`	`15`
`-`	Resta	`10 - 5`	`5`
`*`	Multiplicación	`10 * 5`	`50`
`/`	División	`10 / 4`	`2` (división entera)
`%`	Módulo (Resto)	`10 % 4`	`2` (resto de `10 / 4`)

Error Común: La División Entera

¡Mucho cuidado con esto! Cuando los dos operandos de una división `/` son de tipo entero (`byte`, `short`, `int`, `long`), Java realiza una **división entera**, lo que significa que descarta por completo cualquier resto decimal. * `10 / 3` da como resultado `3`. * `7 / 2` da como resultado `3`. * `1 / 2` da como resultado `0`.

Este comportamiento es útil en algunos casos, pero una fuente de errores muy común si lo que esperas es un resultado decimal. En la próxima semana, aprenderemos la técnica para manejar estas situaciones y obtener el resultado preciso.

(Página 2 de 4)

3.2.- Operadores de Asignación

Se utilizan para asignar un valor a una variable. El más básico es `=`, pero existen operadores combinados que realizan una operación y una asignación en un solo paso, haciendo el código más conciso.

Operador	Ejemplo	Equivalente a	Significado
`=`	`x = 10;`	-	Asigna 10 a `x`.
`+=`	`x += 5;`	`x = x + 5;`	Incrementa `x` en 5.
`-=`	`x -= 5;`	`x = x - 5;`	Decrementa `x` en 5.
`*=`	`x *= 5;`	`x = x * 5;`	Multiplica `x` por 5.

```
int numeroAsistentes = 100;
numeroAsistentes += 5; // Llega un grupo de 5. Ahora numeroAsistentes vale 105.
```

3.3.- Operadores Relacionales (de Comparación)

Comparan dos valores y el resultado de la expresión es siempre un valor booleano (`true` o `false`). Son la base para la toma de decisiones en los programas.

Operador	Nombre	Ejemplo	Resultado si aforo=100, max=100
`==`	Igual a	`aforo == max`	`true`
`!=`	Distinto de	`aforo != max`	`false`
`>`	Mayor que	`aforo > max`	`false`
`<`	Menor que	`aforo < max`	`false`
`>=`	Mayor o igual que	`aforo >= max`	`true`
`<=`	Menor o igual que	`aforo <= max`	`true`

¡El Error Más Famoso de la Programación! (`==` vs `=`)

Un error extremadamente común es confundir el operador de comparación `==` con el de asignación `=`. * `if (estaLleno = true)` **ASIGNA** el valor `true` a `estaLleno`. * `if (estaLleno == true)` **COMPARA** si el valor de `estaLleno` es igual a `true`.

En Java, el primer caso suele dar un error de compilación dentro de un `if`, lo que ayuda a prevenir el bug.

3.4.- Operadores Lógicos

Se utilizan para combinar dos o más expresiones booleanas, permitiéndonos crear condiciones complejas.

Operador	Nombre	Ejemplo	Resultado
`&&`	Y (AND)	`(edad >= 18) && tieneEntrada`	`true` solo si **ambas** condiciones son verdaderas.
`||`	O (OR)	`esVip || tieneDescuento`	`true` si **al menos una** de las condiciones es verdadera.
`!`	NO (NOT)	`!estaAgotado`	Invierte el valor booleano (`true` a `false` y viceversa).

Ejemplo de uso en `EventDEV`: Para obtener un descuento, un usuario debe ser mayor de 65 años O ser estudiante.

```
int edadUsuario = 22;
boolean esEstudiante = true;
boolean tieneDerechoADescuento = (edadUsuario > 65) || esEstudiante;
// (false) || (true)  -->  El resultado de la expresión es true.
```

2.2.4 4.- Precedencia de Operadores

En una expresión con múltiples operadores, como `5 + 10 * 2`, Java no los evalúa simplemente de izquierda a derecha. Al igual que en matemáticas, los operadores tienen un **orden de precedencia** que dicta qué operaciones se realizan antes. La multiplicación y la división tienen mayor precedencia que la suma y la resta. Por lo tanto, en el ejemplo, el resultado es `5 + 20 = 25`.

Orden de Precedencia (simplificado, de mayor a menor): 1. Operadores unarios (`++`, `--`, `!`) 2. Multiplicativos (`*`, `/`, `%`) 3. Aditivos (`+`, `-`) 4. Relacionales (`>`, `<`, `==`, etc.) 5. Lógicos (`&&`, `||`) 6. Asignación (`=`, `+=`, etc.)

La regla de oro: Cuando tengas la más mínima duda o simplemente quieras que tu código sea más claro para otros, **utiliza paréntesis `()`**. Las expresiones dentro de los paréntesis siempre se evalúan primero.

```
int resultado = (5 + 10) * 2; // Ahora el resultado es 30, sin ambigüedad.
```

Para Saber Más: Evaluación de "Cortocircuito"

Los operadores `&&` y `||` son eficientes. En una expresión `A && B`, si `A` es `false`, Java ya sabe que el resultado total será `false`, por lo que **no se molesta en evaluar** `B`. De igual forma, en una expresión `A || B`, si `A` es `true`, Java ya sabe que el resultado total será `true` y **no evalúa** `B`. Esto se llama "evaluación de cortocircuito" y es un concepto importante en la optimización y prevención de errores.

(Página 4 de 4)

2.2.5 RESUMEN DE LA SEMANA 2 Y AUTOEVALUACIÓN

En esta sesión hemos aprendido a manipular los datos, pasando de un almacenamiento estático a la creación de lógica dinámica.

Conceptos Clave: * **Operador:** Un símbolo (`+`, `*`, `>`, `&&`) que realiza una operación sobre operandos. * **Expresión:** Una combinación de operandos y operadores que se evalúa para producir un único valor. * **Clasificación de Operadores:** * **Aritméticos:** Para cálculos matemáticos. Cuidado con la división entera. * **Relacionales:** Para comparaciones que resultan en `true` o `false`. * **Lógicos:** Para combinar expresiones booleanas. * **Precedencia:** El orden en el que se evalúan los operadores. Usa paréntesis para asegurar la claridad y el orden deseado.

Diagrama de Flujo Conceptual:

[DIAGRAMA: Un esquema. "Tengo datos en variables" -> "Necesito calcular/comparar" -> "Elijo los operadores adecuados (`+`, `*`, `>`, `&&`...)" -> "Construyo una expresión (considerando la precedencia y paréntesis)" -> "El resultado es un nuevo valor" -> "Asigno el resultado a otra variable".]

Cita para pensar

"Escribir código es el proceso de traducir la lógica humana, llena de condiciones y cálculos, a un lenguaje estricto que una máquina pueda ejecutar sin ambigüedad. Los operadores son el vocabulario de esa traducción." - Anónimo

Autoevaluación

1. ¿Cuál es el resultado de la expresión `int resultado = 10 % 3 + 5 / 2;`?
   - a) 3
   - b) 3.5
   - c) 1
2. Evalúa la siguiente expresión booleana: `(10 > 5) && !(8 == 4 * 2)`.
3. Si `x` es `10`, ¿qué valor tendrá `x` después de ejecutar `x += 5 * 2;`?

(Respuestas: 1. a) 3 (1 + 2). 2. `false` (`true && !true` -> `true && false`). 3. 20 (`x = 10 + 10`).)

2.2.6 PRÁCTICA - SESIÓN 5

(Página 1 de 2)

2.2.7 La Calculadora de Tickets de Evento

Caso práctico

Laia le pide al equipo que implemente la lógica de cálculo de un ticket de compra en `EventDEV`. "Necesitamos calcular el subtotal, ver si el cliente tiene derecho a un descuento por volumen y finalmente, calcular el total. Es hora de que esos operadores trabajen."

Objetivos de la Sesión

Al finalizar esta práctica, serás capaz de: * Utilizar operadores aritméticos (`*`, `+`) para implementar una lógica de cálculo. * Utilizar operadores relacionales (`>`) para comprobar condiciones. * Utilizar operadores de asignación combinados (`-=`). * Resolver un escenario de depuración con operadores.

Desarrollo Paso a Paso (Parte Guiada)

1. **Crea un nuevo proyecto Java en IntelliJ** llamado `CalculadoraTicket`.
2. **Declara las variables y constantes** dentro del `main`:

```
String nombreProducto = "Entrada General Concierto";
int cantidad = 3;
double precioUnitario = 89.95;
final double UMBRAL_DESCUENTO = 200.0;
final double DESCUENTO = 25.0;
```

3. **Realiza los cálculos:**

```
double subtotal = cantidad * precioUnitario;
boolean aplicaDescuento = subtotal > UMBRAL_DESCUENTO;
double total = subtotal;

// La estructura 'if' la veremos en detalle, pero ejecuta el código
// de dentro solo si la condición es 'true'.
if (aplicaDescuento == true) {
    total -= DESCUENTO; // total = total - DESCUENTO
}
```

4. **Imprime el ticket por consola:**

```
System.out.println("--- TICKET DE COMPRA ---");
System.out.println("Producto: " + nombreProducto);
System.out.println("Subtotal: " + subtotal + " EUR");
System.out.println("¿Tiene descuento?: " + aplicaDescuento);
System.out.println("TOTAL A PAGAR: " + total + " EUR");
```

5. **Ejecuta el programa (`Shift + F10`)** y comprueba el resultado. Con los datos iniciales, el descuento debería aplicarse.

(Página 2 de 2)

¡Ahora tú! (Retos)

Reto 1: Calculadora de IVA (Consolidación) Crea un nuevo programa que, dado un precio base, calcule y muestre el IVA y el precio final. 1. Crea una constante `double` para el `IVA` (usa `0.21`). 2. Crea una variable `double` para el `precioBase` (ej: `120.0`). 3. Calcula el `valorIva` (el importe del impuesto). 4. Calcula el `precioFinal` (precio base + IVA). 5. Imprime un resumen claro: "Precio Base: X, IVA: Y, Total: Z".

Reto 2: Repartidor de Botín (Ampliación) Imagina un videojuego donde 5 aventureros consiguen un botín de 103 monedas de oro. El oro se reparte a partes iguales entre ellos. 1. Crea variables `int` para `monedasOro` y `numeroAventureros`. 2. Calcula cuántas monedas le tocan a cada aventurero. 3. Calcula cuántas monedas **sobran** después del reparto. 4. Imprime: "Cada aventurero recibe X monedas. Sobran Y monedas para el tesoro del gremio." *Pista: Necesitarás los operadores `/` y `%`.*

Reto 3: Depuración de Código Copia el siguiente código en un nuevo proyecto. Tiene un error lógico sutil. El subtotal es 150, por lo que el descuento no debería aplicarse y el total debería ser 150. Sin embargo, al ejecutarlo, el total es 135. Encuentra y corrige el bug.

```
// CÓDIGO CON ERROR
public class DepuracionOperadores {
    public static void main(String[] args) {
        double subtotal = 150.0;
        double descuento = 15.0;
        double total = subtotal;

        // Se quiere aplicar el descuento SOLO si el subtotal es MAYOR a 200
        if (subtotal = 200.0) { // ¡Cuidado aquí!
            total -= descuento;
        }

        System.out.println("Total a pagar: " + total);
```

```
    }
}
```*Pista: Relee la sección de "Errores Comunes" de la teoría.*

> #### **Autoevaluación**
> *   ¿He utilizado operadores aritméticos para calcular el subtotal? **[SÍ / NO]**
> *   ¿He resuelto el Reto 2 usando correctamente los operadores de división y módulo? **[SÍ / NO]**
> *   ¿He encontrado y corregido el bug de asignación vs. comparación del Reto 3? **[SÍ / NO]**

***
PRÁCTICA - SESIÓN 6

(Página 1 de 2)

El Validador de Acceso al Evento

Caso práctico
"Equipo, una funcionalidad clave de `EventDEV` es validar si un usuario puede acceder a un evento o a zonas especiales", dice Laia. "Vamos a crear un simulador que, dados los datos de un usuario, nos diga si puede entrar a un evento para mayores de 18 y si tiene acceso a la zona VIP."

Objetivos de la Sesión
*   Declarar y utilizar variables booleanas para representar estados.
*   Construir expresiones complejas utilizando operadores lógicos `&&` (Y) y `||` (O).
*   Imprimir los resultados de las validaciones lógicas y experimentar con ellos.

Desarrollo Paso a Paso (Parte Guiada)

1. **Crea un nuevo proyecto Java** llamado `ValidadorAcceso`.
2. **Declara las variables de estado** de un usuario de prueba dentro del `main`:
    ```java
    int edadUsuario = 25;
    boolean tieneEntrada = true;
    boolean esVip = false;
    boolean autorizacionPaterna = false;
    ```
3. **Crea las expresiones de validación** usando operadores lógicos:
    ```java
    // Para entrar al evento, debe tener entrada Y ser mayor o igual de 18.
    boolean puedeEntrar = tieneEntrada && (edadUsuario >= 18);

    // Para acceder a la zona VIP, primero debe poder entrar Y además ser VIP.
    boolean tieneAccesoZonaVip = puedeEntrar && esVip;
    ```
4. **Imprime los resultados** de las validaciones:
    ```java
    System.out.println("--- VALIDACIÓN DE ACCESO ---");
    System.out.println("Edad del usuario: " + edadUsuario);
    System.out.println("¿Tiene entrada?: " + tieneEntrada);
    System.out.println("¿Es VIP?: " + esVip);
    System.out.println("----------------------------");
    System.out.println("¿Puede entrar al evento?: " + puedeEntrar);
    System.out.println("¿Tiene acceso a la zona VIP?: " + tieneAccesoZonaVip);
    ```
5. **Ejecuta el programa** (`Shift + F10`)** y comprueba el resultado.

Experimenta y Aprende
Cambia los valores de las variables iniciales una por una y vuelve a ejecutar el programa para ver cómo afectan al resultado final.
*   Cambia `edadUsuario` a `17`. ¿Qué resultado esperas? ¿Y cuál obtienes?
*   Con `edadUsuario = 17`, cambia `tieneEntrada` a `false`. ¿Cambia el resultado de `puedeEntrar`? ¿Por qué?
*   Vuelve a poner `edadUsuario = 25` y `tieneEntrada = true`. Ahora cambia `esVip` a `true`. ¿Qué resultado esperas para `tieneAccesoZonaVip`?

(Página 2 de 2)

¡Ahora tú! (Retos)

Reto 1: Acceso para Menores (Consolidación)
La política de acceso cambia. Ahora, un menor de edad (menos de 18) también puede entrar si tiene entrada **Y** cuenta con autorización paterna.
1. Modifica la expresión de la variable `puedeEntrar`.
2. La nueva lógica es: `(tieneEntrada Y (es mayor de 18 O tiene autorización paterna))`.
3. Necesitarás combinar los operadores `&&` y `||`, y usar paréntesis `()` para asegurar el orden correcto de las operaciones.
4. Prueba tu nueva lógica con diferentes combinaciones de valores (un mayor de edad, un menor con autorización, un menor sin autorización).

Reto 2: Política de Descuentos Compleja (Ampliación)
Un usuario de `EventDEV` tiene un 10% de descuento en la tienda del evento si cumple **alguna** de las siguientes condiciones:
*   Es VIP.
*   Es mayor de 65 años.
*   Es menor de 18 años y tiene entrada.

1. Crea un nuevo programa.
2. Declara variables para `esVip` (`boolean`), `edad` (`int`) y `tieneEntrada` (`boolean`).
3. Crea una única variable booleana `aplicaDescuentoTienda` que contenga toda la lógica.
4. Prueba el programa con varios perfiles de usuario para verificar que la lógica es correcta:
    *   Un VIP de 30 años.
    *   Un no-VIP de 70 años.
    *   Un no-VIP de 16 años con entrada.
    *   Un no-VIP de 16 años sin entrada.
    *   Un no-VIP de 40 años.

> #### **Autoevaluación**
> *   ¿He utilizado el operador `&&` y `||` para combinar condiciones? **[SÍ / NO]**
> *   ¿Entiendo la importancia de los paréntesis para agrupar expresiones lógicas? **[SÍ / NO]**
> *   ¿He resuelto el Reto 1, modificando la lógica de acceso para menores? **[SÍ / NO]**
> *   ¿He sido capaz de traducir la política de descuentos del Reto 2 a una única expresión lógica? **[SÍ / NO]**

```
***
## **Semana 3: El Arte de la Conversión de Tipos**

### **PLAN DE TRABAJO (8 horas)**
*   **Sesión 7 (Teoría, 2 horas):** Conversión de tipo implícita (promoción) y explícita (casting).
*   **Sesión 8 (Práctica, 3 horas):** El Laboratorio de Conversiones.
*   **Sesión 9 (Práctica, 3 horas):** Práctica Final: Resumen Financiero del Evento.

***
### **TEORÍA - SESIÓN 7**

**(Página 1 de 4)**

### **4.- Conversión de Tipos**

En la semana anterior, vimos cómo funcionan los operadores, pero dejamos una pregunta importante en el aire: ¿qué pasa cuando operamos con tipos de datos diferentes, como un `int` y un `double`? ¿Y cómo solucionamos el problema de la división entera? La respuesta a ambas preguntas está en la **conversión de tipos**.

Java, al ser un lenguaje fuertemente tipado, es muy estricto con los tipos de datos. Por norma general, no puedes, por ejemplo, meter un dato de tipo `double` en una "caja" (variable) de tipo `int` sin más. Sin embargo, Java proporciona mecanismos para manejar estas situaciones de forma segura.

#### **Caso práctico**
María está frustrada. "Laia, estoy intentando calcular la valoración media de un evento. Han votado 3 personas (`int`) con un total de 10 estrellas (`int`). Mi programa me dice que la media es 3.0, ¡y no 3.33! He guardado el resultado en una variable `double`, pero no funciona."
Laia sonríe. "Un rito de iniciación para todo programador en Java. Has caído en la trampa de la división entera. Tu programa está dividiendo dos números enteros, y el resultado es un entero (`3`). Es después de esa operación cuando se convierte a `double` (`3.0`), pero ya es demasiado tarde. Tienes que 'forzar' a que la división se haga con decimales desde el principio. Hoy aprenderás la técnica para hacerlo: el casting."

#### **4.1.- Conversión Implícita (Promoción)**
Este tipo de conversión es automática y segura. Ocurre cuando asignamos un tipo de dato "pequeño" a uno "grande". Se llama **promoción** porque el tipo más pequeño se "promociona" al más grande. Como no hay riesgo de perder información, Java lo hace por nosotros sin quejarse.

Es como verter el contenido de un vaso pequeño en una jarra grande; siempre cabrá.

```java
int numVotos = 150;
double numeroDeVotosDouble = numVotos; // CORRECTO: numVotos (int) se promociona a double.
                                       // numeroDeVotosDouble valdrá 150.0
```

Esto también ocurre dentro de las expresiones:

```java
int numAsistentes = 100;
double precioEntrada = 75.50;

// Antes de multiplicar, Java promociona numAsistentes a 100.0 (double).
// Luego multiplica dos doubles: 100.0 * 75.50.
double recaudacion = numAsistentes * precioEntrada;
```

La jerarquía de promoción para evitar pérdida de datos es la siguiente: `byte` -> `short` -> `int` -> `long` -> `float` -> `double` Un `char` también puede ser promocionado a `int` (y superiores), usando su valor numérico Unicode.

(Página 2 de 4)

2.2.8 4.2.- Conversión Explícita (Casting)

Este tipo de conversión es manual y potencialmente peligrosa. Ocurre cuando queremos forzar la conversión de un tipo "grande" a uno "pequeño". Como aquí **sí hay riesgo de perder información**, Java no lo hace automáticamente. Nos obliga a decirle explícitamente: "Sé lo que estoy haciendo y asumo las consecuencias".

Es como intentar verter el contenido de una jarra grande en un vaso pequeño; es muy probable que parte del líquido se derrame.

La sintaxis para este "molde" o **casting** es poner el tipo de destino entre paréntesis antes de la variable o expresión a convertir.

`(tipo_destino) valor_a_convertir;`

El Error de "Pérdida de Precisión"

Si intentas hacer una conversión de grande a pequeño sin un casting, el IDE y el compilador te darán un error.

```java
double precioConDecimales = 75.99;

// La siguiente línea da un error de compilación:
// "incompatible types: possible lossy conversion from double to int"
// int precioEntero = precioConDecimales;
```
```

IntelliJ IDEA subraya el código en rojo y te avisa del peligro.

Solucionándolo con Casting

Para que el código compile, debemos añadir el casting explícito. Al hacerlo, aceptamos la posible pérdida de datos.

```
double precioConDecimales = 75.99;

// Añadimos (int) para forzar la conversión.
// Java TRUNCA (corta) la parte decimal. No redondea.
int precioEntero = (int) precioConDecimales; // precioEntero ahora valdrá 75
```**Otros ejemplos de casting:**
```java
long numeroGrande = 100L;
int numeroPequeño = (int) numeroGrande; // Casting de long a int

float valorFloat = 3.14F;
short valorShort = (short) valorFloat; // valorShort valdrá 3
```

Para Saber Más: El Peligro del Desbordamiento (Overflow)

El casting de un número grande a un tipo pequeño puede dar resultados inesperados si el número no cabe. `int numeroMuyGrande = 300; byte numeroByte = (byte) numeroMuyGrande;` El rango de `byte` es -128 a 127. `300` no cabe. `numeroByte` no valdrá `127` ni dará un error en ejecución. Tomará un valor extraño (`44` en este caso) debido a cómo se representan los números en binario. Es un bug difícil de detectar. ¡Usa el casting con cuidado!

(Página 3 de 4)

2.2.9 4.3.- Aplicación Práctica: Resolviendo la División Entera

Ahora tenemos la herramienta para resolver el problema que planteamos en la Semana 2 y que sufría María en el caso práctico.

Recordemos el problema:

```
int totalEstrellas = 10;
int numeroVotos = 3;

// Java evalúa '10 / 3' como una división entera. El resultado es 3.
// Luego, el 3 (int) se promociona a 3.0 (double) para asignarlo a la variable.
// Pero ya es demasiado tarde, la precisión se ha perdido.
double mediaIncorrecta = totalEstrellas / numeroVotos; // El resultado es 3.0
```

La solución es usar un casting para asegurarnos de que la división se realice con números de coma flotante desde el principio. Solo necesitamos convertir **uno** de los operandos.

Solución correcta:

```
int totalEstrellas = 10;
int numeroVotos = 3;

// 1. Casteamos totalEstrellas a double. El valor se convierte en 10.0.
// 2. La expresión se convierte en 10.0 / 3.
// 3. Como un operando es double, Java promociona el otro (3 se convierte en 3.0).
// 4. Se realiza una división de doubles: 10.0 / 3.0.
// 5. El resultado (3.333...) se asigna a la variable.
double mediaCorrecta = (double) totalEstrellas / numeroVotos;
```

También funcionaría casteando el segundo operando, o ambos: `double mediaAlternativa1 = totalEstrellas / (double) numeroVotos;` `double mediaAlternativa2 = (double) totalEstrellas / (double) numeroVotos;`

La clave es que, en el momento de la división, al menos uno de los dos operandos sea de tipo `double` o `float`.

Cita para pensar

"La programación es el arte de ser explícito. La conversión implícita es una comodidad que nos ofrece el lenguaje, pero la conversión explícita es una declaración de intenciones del programador." - Anónimo

---

(Página 4 de 4)

2.2.10 RESUMEN DE LA SEMANA 3 Y AUTOEVALUACIÓN

En esta sesión hemos aprendido cómo Java maneja las operaciones entre diferentes tipos de datos, un concepto fundamental para escribir código robusto y sin errores de cálculo.

Conceptos Clave: * **Conversión Implícita (Promoción):** * Automática y segura. * Ocurre de un tipo de dato pequeño a uno grande. * Java lo hace por nosotros (`int` a `double`). * **Conversión Explícita (Casting):** * Manual y potencialmente peligrosa (pérdida de datos). * Ocurre de un tipo de dato grande a uno pequeño. * Se usa la sintaxis `(tipo_destino)`. * Es **esencial** para solucionar el problema de la división entera. * **Truncamiento:** Al castear de `double` a `int`, la parte decimal se corta, no se redondea. `(int) 9.9` es `9`.

Diagrama de Decisión:

[DIAGRAMA: Un esquema. "¿Voy a asignar un valor a una variable de otro tipo?" -> "¿Es una conversión de 'pequeño' a 'grande' (ej. int a double)?" (Sí -> Conversión Implícita, Java se encarga. No -> Es de 'grande' a 'pequeño' (ej. double a int)) -> "Necesito Conversión Explícita (Casting). ¿Estoy seguro de que la posible pérdida de datos es aceptable?" (Sí -> Escribo `(tipo)valor`. No -> Reconsidero el diseño, quizás la variable destino debería ser de un tipo más grande).]

Autoevaluación

1. ¿Qué valor contendrá la variable `resultado` tras ejecutar `int resultado = (int) 2.7 + 5;`?
   - a) 7
   - b) 7.7
   - c) 8
2. ¿Cuál de las siguientes líneas de código **no** producirá un error de compilación?
   - a) `int x = 3.0;`
   - b) `float y = 3.0;`
   - c) `double z = 3;`
3. Tienes `int a = 5;` y `int b = 2;`. ¿Qué expresión escribirías para que una variable `double resultado` contenga el valor `2.5`?

(Respuestas: 1. a) 7 (`2 + 5`). 2. c) (es una promoción implícita). 3. `double resultado = (double) a / b;` o `double resultado = a / (double) b;`)

2.2.11 PRÁCTICA - SESIÓN 8

(Página 1 de 2)

2.2.12 El Laboratorio de Conversiones en EventDEV

Caso práctico

"Equipo, hoy vamos a experimentar", dice Laia. "Quiero que veáis con vuestros propios ojos cómo Java maneja los tipos, cuándo os ayuda automáticamente y cuándo os obliga a tomar una decisión explícita con un casting. Vamos a reproducir el problema de la media de valoraciones de María y a solucionarlo."

Objetivos de la Sesión

- Observar una conversión de tipo implícita en una operación.
- Provocar y entender el error de compilación por "pérdida de precisión".
- Utilizar el casting explícito para convertir tipos y resolver el problema de la división entera.

Desarrollo Paso a Paso (Parte Guiada)

1. **Crea un nuevo proyecto Java** llamado `LaboratorioConversiones`.

2. **Paso 1: Conversión implícita.** Escribe y ejecuta este código:

```
int numAsistentes = 150;
double gastosGestion = 2.50;
double totalOperacion = numAsistentes * gastosGestion;
System.out.println("Total (implícito): " + totalOperacion);
```

3. **Paso 2: Provoca el error.** Intenta asignar un `double` a un `int`:

```
double recaudacion = 54321.99;
// La siguiente línea da error, obsérvalo en IntelliJ.
// int recaudacionEntera = recaudacion;
```

4. **Paso 3: Soluciónalo con casting.** Corrige la línea anterior:

```
int recaudacionEntera = (int) recaudacion;
System.out.println("Recaudación entera (casting): " + recaudacionEntera);
```

(Página 2 de 2)

1. **Paso 4: El problema de la división entera.**

```
int totalEstrellas = 10;
int numeroVotos = 3;
double mediaIncorrecta = totalEstrellas / numeroVotos;
System.out.println("Media incorrecta: " + mediaIncorrecta);
```

2. **Paso 5: La solución.**

```
double mediaCorrecta = (double) totalEstrellas / numeroVotos;
System.out.println("Media correcta: " + mediaCorrecta);
```

3. **Ejecuta el programa (`Shift + F10`)** y analiza cada una de las salidas en la consola, especialmente la diferencia entre el paso 4 y el 5.

¡Ahora tú! (Retos)

Reto 1: Porcentaje de Ocupación (Consolidación) Tienes dos variables `int`: `entradasVendidas = 85` y `aforoTotal = 200`. Calcula y muestra el **porcentaje de ocupación** del evento con decimales. *Pista: La fórmula es `(entradasVendidas / aforoTotal) * 100`, pero si no usas un casting, la división `85 / 200` dará `0`.*

Reto 2: Conversión de Caracteres a Números (Ampliación) Cada `char` en Java tiene un valor numérico Unicode asociado. 1. Declara un `char letra = 'A';`. 2. Declara un `int codigoLetra` y asígnale el valor de `letra`. ¿Necesitas un casting? (No, es una promoción implícita). Imprime el código numérico. 3. Declara un `int codigoSiguiente = codigoLetra + 1;`. 4. Declara un `char letraSiguiente` y asígnale el valor de `codigoSiguiente`. ¿Necesitas un casting? (Sí, de `int` a `char`). Imprime la letra siguiente. El resultado debería ser 'B'.

Autoevaluación

- ¿Entiendo cuándo Java convierte tipos automáticamente? **[SÍ / NO]**
- ¿He resuelto el Reto 1, evitando la división entera con un casting? **[SÍ / NO]**
- ¿He completado el Reto 2 y entiendo la relación numérica de los `char`? **[SÍ / NO]**

2.2.13 PRÁCTICA - SESIÓN 9

(Página 1 de 2)

2.2.14 **Práctica Final: Resumen Financiero del Evento**

Caso práctico

"Muy bien, equipo, es la hora de la verdad", anuncia Laia. "Vamos a crear un programa final para esta unidad que junte todo lo que hemos aprendido: variables, constantes, todos los tipos de operadores y conversiones de tipo. Crearemos un resumen financiero para un evento de `EventDEV`."

Objetivos de la Sesión

- Integrar todos los conceptos de la Unidad 1 en un único programa coherente.
- Resolver un problema práctico que requiere el uso combinado de variables, constantes, operadores y conversiones.

Desarrollo Paso a Paso (Parte Guiada)

1. **Crea un nuevo proyecto Java** llamado `ResumenFinanciero`.
2. **Declara las constantes y variables** de partida:

```
// --- CONSTANTES DE NEGOCIO ---
final double PRECIO_ENTRADA_GENERAL = 50.0;
final double PRECIO_ENTRADA_VIP = 150.0;
final double IVA = 0.21;

// --- DATOS DEL EVENTO ---
int entradasGeneralesVendidas = 200;
int entradasVipVendidas = 50;
```3.  **Realiza los cálculos:**
```java
// --- CÁLCULOS FINANCIEROS ---
// 1. Total de entradas vendidas
int totalEntradas = entradasGeneralesVendidas + entradasVipVendidas;

// 2. Recaudación total sin impuestos (bruta)
double recaudacionBruta = (entradasGeneralesVendidas * PRECIO_ENTRADA_GENERAL) + (entradasVipVendidas * PRECIO_ENTRADA_VIP);

// 3. Impuestos a pagar
double impuestos = recaudacionBruta * IVA;

// 4. Recaudación neta (después de impuestos)
double recaudacionNeta = recaudacionBruta - impuestos;

// 5. Precio medio por entrada (¡OJO CON LA DIVISIÓN ENTERA!)
double precioMedioEntrada = recaudacionBruta / totalEntradas;
```

(Página 2 de 2)

1. **Imprime el informe financiero** por consola:

```
// --- INFORME FINANCIERO DEL EVENTO ---
System.out.println("-------------------------------------");
System.out.println("Total Entradas Vendidas: " + totalEntradas);
System.out.println("Recaudación Bruta: " + recaudacionBruta + " EUR");
System.out.println("Impuestos (IVA " + (IVA * 100) + "%): " + impuestos + " EUR");
System.out.println("Recaudación Neta: " + recaudacionNeta + " EUR");
System.out.println("Precio Medio por Entrada: " + precioMedioEntrada + " EUR");
System.out.println("-------------------------------------");
```

2. **Ejecuta el programa (`Shift + F10`)** y verifica que todos los cálculos son correctos.

¡Ahora tú! (Retos)

Reto 1: Comisión del Organizador (Consolidación) Añade una nueva funcionalidad al informe. La empresa organizadora (un tercero) se lleva una comisión del 5% sobre la **recaudación neta**. 1. Declara una nueva constante `COMISION_ORGANIZADOR` con valor `0.05`. 2. Calcula el `beneficioOrganizador` (el importe de la comisión). 3. Calcula el `beneficioFinal` para Azahar Tech (recaudación neta menos la comisión del organizador). 4. Añade estas dos nuevas líneas al informe impreso.

Reto 2: Desglose de Entradas (Ampliación) Añade al informe el porcentaje que representa cada tipo de entrada sobre el total de entradas vendidas. 1. Calcula el `porcentajeEntradasGenerales`. 2. Calcula el `porcentajeEntradasVip`. 3. Añade estas dos líneas al

informe, mostrando el resultado con decimales. *Pista: De nuevo, ten cuidado con la división entera.* `(entradasGeneralesVendidas / totalEntradas)` *dará 0. Necesitarás un casting.*

Reto 3: Depuración de Lógica Imagina que el cálculo de los impuestos está mal en el código original. En lugar de `recaudacionBruta * IVA`, alguien ha escrito `recaudacionBruta / IVA`. Ejecuta el programa con ese error. ¿El resultado es muy diferente? ¿Es fácil ver que está mal? Este ejercicio demuestra por qué usar nombres de variables claros (`impuestos`) es crucial: te ayuda a verificar si la operación que estás haciendo tiene sentido lógico. Revierte el error y déjalo funcionando correctamente.

Autoevaluación

- ¿He utilizado variables y constantes de forma apropiada? **[SÍ / NO]**
- ¿He combinado operadores aritméticos para realizar cálculos complejos? **[SÍ / NO]**
- ¿He manejado correctamente la división para el precio medio y los porcentajes, evitando la división entera? **[SÍ / NO]**
- ¿Mi programa final se ejecuta y produce un informe coherente y correcto, incluyendo los retos? **[SÍ / NO]**

3. Unidad 2: Tema avanzado

Aquí iría el contenido de la **Unidad 2**.